Java

- Scanner

Scanner

- ```
  final class java.util.Scanner
  implements Iterator<String>, Closeable
  ```
- java.util.Scanner kann:
  - Zeichenketten in Tokens zerlegen
  - Dateien zeilenweise einlesen
- Delimiter – kann regulärer Ausdruck sein
  - flexibler als String – Tokenizer: (Trennzeichen: einzelne Zeichen)
  ```

Scanner – Konstruktoren (Quellen)

- **Scanner - Konstruktoren**
- Zeichenfolgen aus unterschiedlichen Quellen:
  - String
  - Datenstrom – Einlesen von Kommandozeile mit System.in
  - File – Objekt
  - NIO – Objekte

Scanner– Konstruktoren (Quellen)

- **Scanner - Konstruktoren**
  - Objekt vom Typ Closeable (z.B. Writer) muss mit close() geschlossen werden, String nicht nötig, File schließt Scanner selbstständig
  - java.io.Closeable (Interface) – mit Methode close()

Einlesen einer Datei

- **Scanner: Zeilenweise Einlesen einer Datei**
  - hasNextLine() und nextLine(): zeilenweise Auslesen

```
import java.io.*;
import java.util.Scanner;

public class ReadAllLines
{
  public static void main( String[] args ) throws FileNotFoundException
  {
    Scanner scanner = new Scanner( new File("EastOfJava.txt") );
    while ( scanner.hasNextLine() )
      System.out.println( scanner.nextLine() );
    scanner.close();
  }
}
```


Einlesen einer Datei

- **Scanner: Zeilenweise Einlesen einer Datei**
  - kann eine Ausnahme auslösen
    (Throws FileNotFoundException)
  - hasNextLine()
    true, wenn eine nächste Zeile gelesen werden kann
  - nextLine()
    liefert nächste Zeile

Einlesen von der Standardeingabe

```
public static void main( String[] args )
  {
    System.out.println("Bitte geben Sie eine Zeichenfolge ein");
    Scanner scanner = new Scanner( System.in );
    String s = scanner.next();
    System.out.println(s);
```

- next()-Methode: String als Rückgabe
- next<Typ>()-Methoden:
  lesen nächstes Token ein und konvertieren in gewünschtes Format, etwa in ein double bei nextDouble()
- hasNext<Typ>()- Methoden: lässt sich erfragen, ob weiteres Token von diesem Typ folgt

Einlesen von der Standardeingabe

```
Scanner scanner = new Scanner( "tutego 12 1973 12,03 "
        + "True 123456789000" );
System.out.println( scanner.hasNext() );        // true
System.out.println( scanner.next() );           // tutego
System.out.println( scanner.hasNextByte() );    // true
System.out.println( scanner.nextByte() );       // 12
System.out.println( scanner.hasNextInt() );     // true
System.out.println( scanner.nextInt() );        // 1973
System.out.println( scanner.hasNextDouble() );  // true
System.out.println( scanner.nextDouble() );     // 12.03
System.out.println( scanner.hasNextBoolean() ); // true
System.out.println( scanner.nextBoolean() );    // true
System.out.println( scanner.hasNextLong() );    // true
System.out.println( scanner.nextLong() );       // 123456789000
System.out.println( scanner.hasNext() );        // false
```

Einlesen von der Standardeingabe

- Scanner skip(Pattern pattern)
- Scanner skip(String pattern)
  - überspringt uninteressante Tokens
  - Delimiter werden nicht beachtet
- useRadix(int) ändert die Basis für Zahlen und radix() erfragt sie

Einlesen von der Standardeingabe

- useDelimiter() setzt für Filter-Vorgänge den Delimiter
  - next(String): Trenner setzen
  - next(Pattern): Trennmuster angeben
  - hasNext(String) bzw. hasNext(Pattern) liefern true, wenn das nächste Token dem Muster entspricht

```
String text = "Hänsel-und-Gretel\ngingen-durch-den-Wald";
Scanner scanner = new Scanner( text ).useDelimiter( "-" );
System.out.println( scanner.findInLine( "Wald" ) ); // null
System.out.println( scanner.findInLine( "ete" ) );  // "ete"
System.out.println( scanner.next() );               // "l" "gingen"
System.out.println( scanner.next() );               // "durch"
```

Einlesen von der Standardeingabe

- **Landessprachen**
- Mit passendem Locale-Objekt erkennt der Scanner bei nextDouble() auch Fließkommazahlen mit Komma, etwa "12,34":

```
Scanner scanner = new Scanner( "12,34" ).useLocale( Locale.GERMAN );
System.out.println( scanner.nextDouble() );   // 12.34
```


Einlesen von der Standardeingabe

- **Landessprachen**
- bei deutschsprachigem Betriebssystem auch ohne useLocale(Locale.GERMAN)
- Grund: Scanner setzt das Locale vorher auf Locale.getDefault()
- (bei Deutschen Betriebssystem Locale.GERMAN)
- in englischer Schreibweise angegebene Zahl wie 12.34 wird nicht erkannt und der Scanner meldet eine java.util.InputMismatchException